ก่อนจะถึงโอโอพี

บุญเลิศ เอี่ยมทัศนา

ในช่วงระยะเวลาสองสามปีมานี้ ในวงการคอมพิวเตอร์โดยเฉพาะในสหรัฐอเมริกา มีการกล่าวถึงการเขียนโปรแกรมในแบบโอโอพี (OOP : Object Oriented Programming) กันอย่างกว้างขวาง ทำให้บริษัทบอร์แลนด์และบริษัทไมโครซอฟต์ ทนไม่ไหวต้องทำ Turbo Pascal 5.5 กับ Quick Pascal เพื่อให้เขียนโปรแกรมในแบบโอโอพีบนเครื่องไมโครคอมพิวเตอร์ได้ออกมาแข่งขันกัน

สำหรับเรื่องราวของ Turbo Pascal 5.5 ผมได้นำมากล่าวแล้วในเรื่อง "ทักทายกับเทอร์โบปาสคาล 5.5" (ไมโครฯ ฉบับที่ 52) ส่วนของ Quick Pascal ต้องขอให้ท่านที่ใช้ซอฟต์แวร์นี้อยู่เขียนมาเล่าสู่กันฟังบ้างครับ

โปรแกรมในแบบโอโอพี

ความจริงแล้วเรื่องราวของการเขียนโปรแกรมในแบบโอโอพี ซึ่งเป็นการเขียนโปรแกรมในแนวใหม่ยังใหม่อยู่มากในวงการคอมพิวเตอร์ จนอาจกล่าวได้ว่า สำนักต่าง ๆ ยังกล่าวถึงเรื่องนี้ไม่ตรงกันทีเดียว เช่น

- สำนักหนึ่งอาจกล่าวว่า การเขียนโปรแกรมในแบบนี้เป็นการมองโปรแกรมแต่ละโมดูลเป็นวัตถุ (object) หนึ่ง ๆ ดังเช่นที่มองวัตถุรอบตัวเราเป็น คน รถยนต์ ต้นไม้ บ้าน เป็นสิ่ง ๆ ไป ไม่มองในรายละเอียด

- อีกสำนักหนึ่งอาจกล่าวว่า เหมือนกับการกำกับการแสดงละครซึ่งตัวละครแต่ละตัวจะมีบุคลิก (ข้อมูล) และบทบาท (โปรแกรมย่อย) เป็นของตนเอง ผู้กำกับเพียงแต่กำกับการให้ดำเนินไปตามท้องเรื่องเท่านั้น (ผมจะยกตัวอย่างตามแนวของสำนักนี้ต่อไป)

- อีกสำนักหนึ่งอาจเน้นในเรื่องของวัตถุ (object) ว่าประกอบด้วยสถานะส่วนตัว (private state คือ ข้อมูล) กับกิจกรรม (methods คือโปรแกรมย่อย)

จะเห็นว่ากล่าวกันไปคนละทางแต่ก็สอดคล้องกัน (หากไม่เห็นไม่เข้าใจไม่ต้องห่วงครับจะเข้าใจภายหลัง) จึงเป็นเรื่องที่เรียนรู้ยากเรื่องหนึ่งเข้าทำนองว่า จะต้องรู้เรื่องก่อนจึงจะเข้าใจ เช่นเดียวกับวิชายุทธศาสตร์

ผมจึงนำมาเขียนในแบบให้รู้เรื่องก่อนเช่นกัน ส่วนความเข้าใจเก็บไว้ก่อน และโปรดอย่าสงสัยหรือติดใจในถ้อยคำว่าหมายถึงอะไรแน่ หรือต้องการคำจำกัดความ หรือสงสัยว่าจะมีประโยชน์อะไรนำไปใช้ที่ไหน เขียนโปรแกรมได้อย่างไร เรื่องต่าง ๆ เหล่านี้ขอพักไว้ก่อนครับ

ตัวอย่างแรก

สมมติว่าผมจะเป็นผู้กำกับการแสดงตัวละคร 3 ตัว มี จุดเป็นนางเอก วงกลมเป็นพระเอก และสี่เหลี่ยมเป็นผู้ร้าย ซึ่งผู้เขียนบทให้ผมสื่อความ (message) กับตัวแสดงได้ 3 คำสั่งเท่านั้น คือ

- เตรียมตัว (Init) เพื่อกำหนดตำแหน่งเริ่มแรกของตัวละครบนจอภาพ และอื่น ๆ เช่น รัศมีของวงกลม

M1

```
VAR   actorP : Point;
      actorC : Circle;
BEGIN
  OpenGraph;

  actorP.Init(100,100);  actorC.Init(150,150,50);
  actorP.Show;  actorC.Show;  ReadLn;

  actorP.JumpTo(50,50);  Readln;
  actorC.JumpTo(150,100);  ReadLn;

readln;  CloseGraph
END.
```

- แสดงตัว (Show) เพื่อให้ตัวละคร แสดงตัวคล้ายกับเมื่อเริ่มเปิดม่าน

- กระโดดไปที่ (JumpTo) เพื่อให้ตัวละครกระโดดไปยังตำบลที่ใหม่

ผมจะสามารถเขียนโปรแกรมกำกับการแสดงตัวละครทั้งสามให้กระโดดโลดเต้นไปบนจอภาพได้ดังโปรแกรมหลักในรูป M1 แต่ในโปรแกรมนี้เพื่อให้โปรแกรมสั้นผมได้ตัดผู้ร้ายออก และการแสดงก็ให้กระโดดคนละทีเท่านั้น

(สำหรับคำว่า OpenGraph อยู่ในยูนิต GRP1 รูปที่ 1 และมีคำอื่นอีก ขอให้อ่านคำอธิบายใต้รูปด้วยครับ)

ทางด้านผู้เขียนบท (ซึ่งอาจจะเป็นคนเดียวกันกับผู้กำกับ) ได้เขียนบทไว้ดังแสดงในรูป A1 ถึง A3 ประกอบด้วยบุคลิกและบท มีรายละเอียดดังนี้

ทางด้านภาษา

เทอร์โบปาสคาลได้กำหนดแบบข้อมูลออปเจ็ค (Object) ขึ้นใหม่เพื่อใช้ในการเขียนโปรแกรมแบบโอโอพี มีลักษณะคล้ายเรคอร์ด (Record) มากต่างกันตรงที่ฟิลด์ของออปเจ็คอาจเป็นข้อมูลหรือโปรแกรมย่อยก็ได้ (ให้กำหนดไว้เฉพาะส่วนหัวเท่านั้น) และมีเรื่อง

รูปที่ 1 สำหรับ path ที่กำหนดไว้ใน InitGraph ขอให้เปลี่ยนเป็นไดเรคทอรี ที่เก็บ GRAPH.TPU

```
UNIT  grp1;
Interface
USES      crt, dos, Graph;

PROCEDURE  OpenGraph;

Implementation


PROCEDURE  OpenGraph;
var     ch : Char;  Gd,Gm,res : Integer;
BEGIN
  DetectGraph(Gd,Gm);
  InitGraph(Gd,Gm,'c:\pascal55\disk3\graphs');
  res := GraphResult;
  if  res <> grOK  then
    begin  WriteLn('Graphics error: ',GraphErrorMsg(res));
           Halt(1)  end;
END;


END.
```

A1

```
{ EXP1.PAS  }

PROGRAM  ExampleOne;
uses    crt,graph,grp1;
TYPE    Locate = OBJECT
                   x,y  : Integer;
      procedure   Init(fx,fy : Integer);
      procedure   NewPost(nx,ny : Integer);
      function    GetX : Integer;
      function    GetY : Integer;
              END;

PROCEDURE  Locate.Init(fx,fy : Integer);
begin   x := fx;  y := fy  end;

PROCEDURE  Locate.NewPost(nx,ny : Integer);
begin  x := nx;  y := ny;  end;

FUNCTION  Locate.GetX : Integer;
begin  GetX := x  end;

FUNCTION  Locate.GetY : Integer;
begin  GetY := y  end;
```

A2

```
YPE    Point = OBJECT(Locate)
     procedure  Init(fx,fy : Integer);
     procedure  Hide;
     procedure  Show;
     procedure  JumpTo(nx,ny : Integer);
             END;

ROCEDURE  Point.Init(fx,fy : Integer);
egin  Locate.Init(fx,fy)  end;

ROCEDURE  Point.Hide;
egin  PutPixel(GetX,GetY,GetBkColor)  end;

ROCEDURE  Point.Show;
egin  PutPixel(GetX,GetY,GetColor)  end;

ROCEDURE  Point.JumpTo(nx,ny : Integer);
EGIN
 Hide;  Locate.NewPost(nx,ny);  Show;
ND;
```

A3

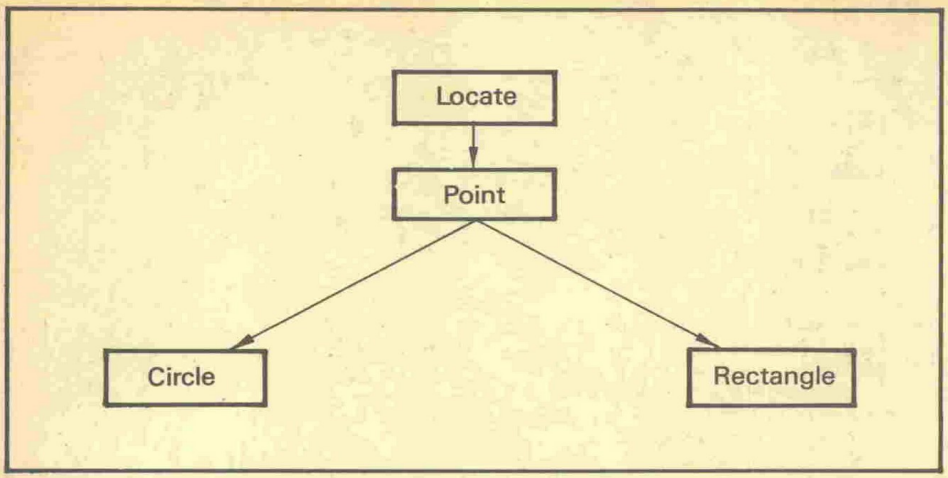
```
TYPE Circle = OBJECT(Point)
                r : Integer;
      procedure  Init(fx,fy,fr : Integer);
      procedure  Hide;
      procedure  Show;
      procedure  JumpTo(nx,ny : Integer);
                 END;

PROCEDURE  Circle.Init(fx,fy,fr : Integer);
begin  Point.Init(fx,fy);  r := fr  end;

PROCEDURE  Circle.Hide;
VAR  tem : Integer;
begin  tem := GetColor;  SetColor(GetBkColor);
       Graph.Circle(GetX,GetY,r);  SetColor(tem)  end;

PROCEDURE  Circle.Show;
begin  Graph.Circle(GetX,GetY,r)   end;

PROCEDURE  Circle.JumpTo(nx,ny : Integer);
begin  Hide;  Point.NewPost(nx,ny);  Show  end;
```



รูปที่ 2 แสดงการสืบสกุล

อื่นอีก คือ

• ให้สืบทอดกันได้ ดังเช่น Circle สืบทอดมาจาก Point ซึ่งสืบทอดมาจาก Locate อีกทอดหนึ่ง และเรียก Locate ว่าเป็นบรรพบุรุษ (ancestor) ของ Point กับ Circle ส่วน Point เป็นบรรพบุรุษของ Circle ในทางกลับกัน Circle จะเป็นผู้สืบสกุล (descendant) ของ Point กับ Locate และ Point เป็นผู้สืบสกุลของ Locate ดังแสดงในรูปที่ 2 (ซึ่งได้นำ Rectangle มาร่วมด้วย)

• ฟิลด์ที่เป็นข้อมูลคือบุคลิก ฟิลด์ที่เป็นโปรแกรมย่อยคือบท ดังที่ได้อุปมาเปรียบเทียบมา

• เมื่อกำหนดส่วนหัวของโปรแกรมย่อยที่เป็นบทแล้ว ให้เขียนโปรแกรมย่อยนั้นโดยใช้ชื่อเป็น

ชื่อออปเจ็ค. ชื่อโปรแกรมย่อย

• ผู้สืบสกุลสามารถใช้ข้อมูลและโปรแกรมย่อยของบรรพบุรุษได้ แต่กลับกันไม่ได้ ดังเช่น Point และ Circle เรียกใช้ฟังก์ชัน GetX ใน Locate และมีรายละเอียดอีกคือ

- GetX (รวมทั้งชื่ออื่นอีก) ในที่นี้ไม่ใช่ฟังก์ชัน GetX ในเทอร์โบปาสคาล หากเป็นฟังก์ชัน GetX ใน Locate (ตามหลักภาษาปาสคาล หากกำหนดชื่อโปรแกรมย่อยทับของเดิมจะเป็นโปรแกรมย่อยตัวใหม่)

- การเรียกใช้ดังกล่าวมาในโปรแกรมของผู้สืบสกุลจะใช้ WITH หรือไม่ก็ได้ ตามตัวอย่างในรูป A2 และ A3 เช่นใน Point.Hide เรียกใช้ GetX และ GetY โดยไม่ได้ใช้ WITH ทำให้สะดวกต่อการเขียนโปรแกรมไม่ต้องใช้ WITH ซ้อน WITH เมื่อมีผู้สืบสกุลหลายทอด (คำว่าใช้ WITH ในที่นี้ขอให้หมายรวมทั้งเมื่อใช้ WITH และเมื่อใช้จุด ตามแบบการอ้างถึงฟิลด์ในเรคอร์ด)

- แต่ถ้าตั้งชื่อฟิลด์ซ้ำกัน (อันเป็นจุดประสงค์ของการเขียนโปรแกรมในแบบโอโอพี ดังจะกล่าวต่อไป) หากจะหมายถึงฟิลด์ในบรรพบุรุษต้องใช้ WITH ดังที่เห็นในโพรซีเดอร์ Point.Init (รูป A2) คำว่า Locate.Init หมายถึง Init ใน Locate (รวมทั้งคำในยูนิตอื่นด้วย เช่น Graph.Circle หมายถึงโพรซีเดอร์ Circle ในยูนิต Graph ของเทอร์โบปาสคาล)

• นำไปกำหนดตัวแปรคือตัวละครด้วย VAR ตามปกติ ดังที่เห็นในรูป M1 ซึ่งได้กำหนดไว้เพียงแบบข้อมูลละ 1 ตัวละครเท่านั้น

• ที่เขียนโปรแกรมย่อยของออปเจ็คไว้หลังออปเจ็คของมันก็เพื่อให้เห็นเทียบกันเท่านั้น จะกำหนดออปเจ็คทั้งหมดก่อน แล้วเขียนโปรแกรมย่อยของแต่ละออปเจ็คได้เช่นกัน และจำเป็นต้องทำด้วยเมื่อเขียนออปเจ็คไว้ในยูนิตดังจะกล่าวต่อไป

• บุคลิกของแต่ละออปเจ็คเป็นข้อมูลแบบโกลบอล (global) คือจะคงอยู่ตลอดการรันโปรแกรม แต่เป็นโลคอล (local) ของตระกูลนั้น คือจะรู้จักในหมู่ทายาทเท่านั้น

ทางด้านโอโอพี

ในการเขียนโปรแกรมแบบโอโอพีพยายามจะแบ่งผู้เขียนโปรแกรมเป็นระดับ ๆ ตามการสืบทอด เช่น เขียนในระดับ Locate ระดับ Point ระดับ Circle และระดับโปรแกรมหลัก (ซึ่งจะเห็นสมจริงเมื่อโปรแกรมมีขนาดใหญ่) โดยให้รู้เรื่องของโปรแกรมในระดับก่อนหน้าตนน้อยที่สุด ด้วยการ

• รวมบทของตัวละครที่เหมือนกันไว้ในตระกูลเดียวกัน (โปรแกรมตามที่ยกมาเป็นตัวอย่างมีเพียงตระกูลเดียว โปรแกรมอื่นอาจมีหลายตระกูล) จากตัวอย่างที่ยกมานี้ได้แก่ เตรียมตัว (Init) หายตัว (Hide) แสดงตัว (Show) กระโดดไปที่ (JumpTo) และตำแหน่งใหม่ (NewPost)

• ให้ผู้เขียนโปรแกรมในแต่ละระดับเข้าใจความหมายของคำทั้ง 5 นี้เหมือนกัน เช่น

- Init หมายถึงให้เตรียมตัว ถ้าเป็น Point ก็จะเตรียมตำแหน่งโคออร์ดิเนตของจุด ถ้าเป็น Circle ก็จะเตรียมตำแหน่งโคออร์ดิเนตของจุดศูนย์กลางและรัศมี ซึ่งจะสัมพันธ์กับเรื่องของบุคลิกคือข้อมูล ดังจะกล่าวต่อไปด้วย

- Hide หมายถึงหายตัว ซึ่งก็คือให้ลบภาพเดิม แต่วิธีหายตัวของจุดและวงกลมไม่เหมือนกัน ของจุดหายตัวด้วยการแสดงจุดด้วยสีพื้นจอภาพทับจุดเดิม ของวงกลมหายตัวด้วยการวาดวงกลมด้วยสีพื้นจอภาพทับวงกลมเดิม

จากที่กล่าวมานี้จะเห็นว่าผู้เขียนโปรแกรมในระดับหลัง ๆ ไม่จำเป็นต้อง

ทราบว่าผู้เขียนโปรแกรมในระดับก่อนหน้านี้ดำเนินการไว้อย่างไร ขอให้ทราบแต่ความหมายของบทก็เพียงพอแล้ว หากสงสัยขอให้ทดลองเขียนออปเจ็ค Rectangle ตามบทดังกล่าวมาดูครับ

• คัดบทเท่าที่จำเป็นให้ผู้กำกับใช้เป็นสื่อความ คือ Init, Show และ JumpTo ความจริงเพียง 2 บทคือ Init กับ JumpTo ก็พอ เพราะสามารถใช้ JumpTo แทน Show ได้ สำหรับผู้กำกับตามที่ผมกล่าวมานี้ ไม่ตรงกับผู้กำกับละครทีเดียวครับ เพราะผู้กำกับของผมนี้จะเป็นผู้ประพันธ์เรื่องต่อไปด้วย

• ไม่ให้ดำเนินการกับบุคลิกของบรรพบุรุษ ดังเช่นค่า X, Y และ r โดยตรงต้องให้โปรแกรมย่อยของออปเจ็คนั้นดำเนินการให้ ดังที่ได้เขียนเป็นโพรซีเดอร์ NewPosition เพื่อเปลี่ยนค่า x กับ y และฟังก์ชัน GetX กับ GetY เพื่ออ่านค่า x และ y

เรื่องนี้ถึงแม้ในเทอร์โบปาสคาลจะสามารถดำเนินการกับบุคลิกได้โดยตรง แต่ภาษาที่สร้างขึ้นเพื่อเขียนโปรแกรมในแบบโอโอพี เช่น Smalltalk ทำไม่ได้จึงไม่ควรทำ ความหมายก็คือต้องเขียนโปรแกรมย่อยเพื่อดำเนินการกับข้อมูลไว้ในออปเจ็คนั้น ดังเช่นโปรแกรมที่กล่าวมา

เปรียบเทียบ

เมื่อเปรียบเทียบการเขียนโปรแกรมในแบบโอโอพีกับการเขียนโปรแกรมธรรมดา จะเห็นว่าการเขียนโปรแกรมในแบบโอโอพีเหมาะกับการเขียนโปรแกรมขนาดใหญ่ เช่นจะเขียนโปรแกรมให้ผู้ใช้เล่นหมากรุกกับคอมพิวเตอร์ กำหนดตัวแปรเป็นโกลบอลมากมาย ซึ่งผู้เขียนโปรแกรมในระดับผู้กำกับจะต้องคอยพะวงถึงเรื่องนี้ เพราะต้องผ่านค่าไปให้โปรแกรมย่อยทุกครั้ง

แต่ถ้าเขียนในแบบโอโอพี จะทำให้ผู้เขียนโปรแกรมในระดับผู้กำกับไม่ต้องคำนึงถึงข้อมูลเลย และอาจมีคำที่ใช้ในการสื่อความเพียง 3 คำเท่านั้นคือ เตรียมตัว กระโดดไปที่ และเข้าโรง ทำให้สามารถระดมปัญญาในการเขียนโปรแกรมเกมหมากรุกได้อย่างเต็มที่

ก่อนที่จะไปต่อ

คำที่ใช้มานี้ผมกำหนดขึ้นเพื่อให้ "รู้เรื่อง" เร็วขึ้น (แต่ถ้ากลายเป็นทำให้เข้าใจยากขึ้นก็ขออภัยด้วยครับ) และถึงแม้ผมจะเรียนให้ทราบแล้วว่าอย่าพึ่งสนใจในถ้อยคำเหล่านี้ แต่ตอนนี้เรากำลังใกล้โอโอพีเข้าไป จำเป็นต้องนำคำในโอโอพีมาใช้ครับ

ปัญหาของผมก็คือคำพวกนี้เป็นคำที่ตั้งขึ้นใหม่ในภาษาอังกฤษ และแม้แต่ในภาษาอังกฤษเองก็ต้องให้ความหมายกัน บางคำสามารถเขียนทับศัพท์ได้ บางคำมีผู้กำหนดคำในภาษาไทยแล้ว ที่เหลือผมจำต้องกำหนดขึ้นใหม่ (พลาง ๆ ก่อนเท่านั้นครับ จนกว่าจะมีผู้รู้หรือสถาบันใดดำเนินการในเรื่องนี้) สำหรับคำในภาษาอังกฤษผมนำมาจากคู่มือเทอร์โบปาสคาล 5.5 เว้นไว้แต่จะกล่าวว่ามาจากที่อื่น (หนังสือแต่ละเล่มมักจะใช้คำไม่ค่อยเหมือนกัน) คำเหล่านี้ได้แก่

- ออปเจ็ค (Object) มีความหมายเป็น 2 นัยคือ ทางด้านภาษาหมายถึงแบบข้อมูล (เช่นเดียวกับเมื่อกล่าวถึงอะเรย์หรือเรคอร์ด) ทางด้านโอโอพีอาจแปลว่า "วัตถุ" แต่ความหมายกระเดียดไปทางด้าน "สิ่ง" (สิ่งหนึ่ง ๆ) มากกว่า ประกอบด้วย characters กับ behaviors หรือ methods

- บุคลิก มาจาก character ซึ่งผมน่าจะใช้คำว่า "ลักษณะ" แต่ที่อื่นใช้คำว่า private state จึงน่าจะตรงกับคำว่า "บุคลิก" มากกว่า (คือฟิลด์ข้อมูลในออปเจ็ค)

- กิจกรรม มาจาก behaviors หรือ method คือ "บท" ตามที่ผมกล่าวมา แต่มีผู้ใช้คำว่ากิจกรรมไว้แล้ว จึงขอใช้ตามให้เหมือนกันครับ (คือฟิลด์โปรแกรมย่อยในออปเจ็ครวมทั้งตัวโปรแกรมย่อย)

- นิมิต มาจาก Instance ซึ่งมีความหมายในทางทำให้เป็นตัวตนขึ้น (instance = allowcated and initialized) ผมจึงขอใช้คำนี้ (เทียบได้กับตัวแปรของข้อมูลแบบอื่น ๆ)

- บรรพบุรุษ เป็นคำแปลของ ancestor คือออปเจ็คที่มีผู้สืบสกุล เช่น Location และ Point

- ผู้สืบสกุล เป็นคำแปลของ descendant คือออปเจ็คที่สืบทอดมาจากออปเจ็คอื่นอีกทอดหนึ่ง เช่น Point และ Circle

การให้ค่าและผ่านค่านิมิต

หากกำหนดนิมิตและเขียนโปรแกรมย่อยไว้ดังรูปที่ 3

รูปที่ 3

```
VAR al : Location;
    ap : Point;
    ac : Circle;
PROCEDURE Example (any :
Point ) ;
begin end;
```

จะเขียนโปรแกรมให้ค่าและผ่านค่าได้ดังนี้

```
al := ap;
ap := ac;
al := ac;
Example (ap) ;
Example (ac) ;
```

คือสามารถให้ค่าผู้สืบสกุลแก่บรรพบุรุษได้ แต่จะให้ค่าบรรพบุรุษแก่ผู้สืบสกุลไม่ได้ เพราะเมื่อให้ค่าบรรพ-

บุรุษแก่ผู้สืบสกุลจะให้ค่าได้ไม่ครบทุกฟิลด์เป็นการไม่ถูกต้อง แต่เมื่อให้ค่าผู้สืบสกุลแก่บรรพบุรุษจะให้ได้ครบทุกฟิลด์ สำหรับฟิลด์ของผู้สืบสกุลที่เหลือเทียบได้กับว่าถูกตัดทิ้งไป

แบบที่สอง

โปรแกรมตามตัวอย่างแรกเป็นการเขียนโปรแกรมตามแบบโอโอพีแบบหนึ่งเรียกว่า แบบสแตติกส์ (static) และก็เหมาะสำหรับงานในลักษณะหนึ่ง เช่น

หากเขียนโปรแกรมในระดับผู้เขียนบทสำหรับตัวหมากรุกทุกตัวทำเป็นยูนิตและคอมพายล์เป็นไฟล์ .TPU เพื่อให้ผู้เขียนโปรแกรมในระดับผู้กำกับนำไปดำเนินการต่อ (ตามความรู้ ความสามารถทางหมากรุกของผู้กำกับแต่ละคน) เมื่อนำไฟล์ .TPU ออกเผยแพร่จะไม่มีปัญหาใด เพราะไม่มีการเพิ่มจำพวกของตัวหมากรุกอีกแล้ว คงมีแต่ ขุน ม้า เรือ ฯลฯ เท่านั้น

แต่ถ้าเป็นการแสดงภาพแบบอื่น เช่น เขียนโปรแกรมแสดงภาพ วงกลม สี่เหลี่ยม และสามเหลี่ยมไว้ในยูนิต เมื่อนำไฟล์ .TPU ออกเผยแพร่จะเกิดปัญหาทันที เพราะผู้เขียนโปรแกรมในระดับผู้กำกับอาจต้องการภาพแบบอื่นอีก เช่น หกเหลี่ยม แต่ไม่สามารถต่อเติมออปเจ็คได้ เพราะไม่มีไฟล์ .PAS

แต่โอโอพีต้องการให้ต่อเติมได้ เรียกว่า เป็นการเขียนโปรแกรมในแบบเวอร์ชวล (virtual) ในการนี้เทอร์โบ-ปาสคาล 5.5 ได้กำหนดคำสงวนไว้ 2 คำ คือ VIRTUAL กับ CONSTRUCTOR เพื่อใช้ประกอบกัน

สำหรับตัวอย่างขอนำตัวอย่างเดิมมาดำเนินการต่อ โดยจะเขียนออปเจ็ค Locate, Point และ Circle ไว้ในยูนิต PICTØ และจะเขียนออปเจ็ค Rectangle ต่อเติมไว้ในไฟล์เดียวกับโปรแกรมหลักกล่าวคือ

ในรูป B1 เป็นส่วนอินเตอร์เฟสของยูนิต PICTØ เพื่อกำหนดแบบข้อมูลของออปเจ็คต่าง ๆ ขอให้ป้อนไว้ในไฟล์ PICTØ.PAS แล้วลอกโปรแกรมย่อยในรูป A1, A2 และ A3 มาไว้ในส่วน Implementation โดยเปลี่ยนคำว่า PROCEDURE ของ Init ใน Point กับ Circle เป็น CONSTRUCTOR (คือให้ตรงกับแบบข้อมูลที่กำหนดไว้ด้วยเหตุผลที่จะกล่าวต่อไปครับ) และลบโพรซีเดอร์ Circle.JumpTo (คือไม่ต้องลอกมา)

โปรแกรมในรูป B2 เป็นการต่อเติมออปเจ็ค Rectangle ขอให้ป้อนโปรแกรมนี้ต่อด้วยโปรแกรมหลักในรูป M2 ลงไฟล์ EXP2.PAS แล้วทดลองรัน (อย่าลืมคอมพายล์ PICTØ.PAS เป็น PICTØ.TPU ก่อน)

จะปรากฏตัวละคร 3 ตัวคือ จุด วงกลม และสี่เหลี่ยมบนจอภาพ เมื่อกดคีย์ Enter ตัวละครทั้งสามนี้จะกระโดดไปทีละตัว ก็เพียงเพื่อแสดงให้เห็นว่าจะต่อเติมออปเจ็คได้อย่างไรเท่านั้นครับ มีเรื่องที่จะกล่าวถึงต่อไปคือ

B1

```
UNIT   PICT0;
Interface
uses   graph;
TYPE   Locate = OBJECT
                  x,y  : Integer;
     procedure  Init(fx,fy : Integer);
     procedure  NewPost(nx,ny : Integer);
     function   GetX : Integer;
     function   GetY : Integer;
           END;

       Point = OBJECT(Locate)
     constructor  Init(fx,fy : Integer);
     procedure  Hide;  VIRTUAL;
     procedure  Show;  VIRTUAL;
     procedure  JumpTo(nx,ny : Integer);
              END;

       Circle = OBJECT(Point)
                  r : Integer;
     constructor  Init(fx,fy,fr : Integer);
     procedure    Hide; VIRTUAL;
     procedure    Show; VIRTUAL;
              END;
```

B2

```
{ EXP2.PAS  }

PROGRAM  ExampleTwo;
uses   crt,graph,grp1,pict0;

TYPE Rectangle = OBJECT(Point)
                   sx,sy : Integer;
     constructor  Init(fx,fy,px,py : Integer);
     procedure  Hide; VIRTUAL;
     procedure  Show; VIRTUAL;
                END;

CONSTRUCTOR  Rectangle.Init(fx,fy,px,py : Integer);
begin  Point.Init(fx,fy);  sx := px;  sy := py  end;

PROCEDURE  Rectangle.Hide;
VAR  xx,yy,tem : Integer;
begin  tem := GetColor;  SetColor(GetBkColor);
       xx := GetX;  yy := GetY;
       Graph.Rectangle(xx,yy,xx+sx,yy+sy);  SetColor(tem)  end;

PROCEDURE  Rectangle.Show;
VAR  xx,yy : Integer;
begin  xx := GetX;  yy := GetY;
       Graph.Rectangle(xx,yy,xx+sx,yy+sy)  end;
```

```
VAR   actorP : Point;
      actorC : Circle;
      actorR : Rectangle;
BEGIN
  OpenGraph;

  actorP.Init(100,100);  actorC.Init(150,150,50);
  actorR.Init(200,100,50,50);
  actorR.Show;  actorC.Show;  actorP.Show;  readln;

  actorP.JumpTo(50,50);  Readln;
  actorC.JumpTo(150,100);  ReadLn;
  actorR.JumpTo(200,50);

readln;  CloseGraph
END.
```

M2

ทางด้านภาษา

การเขียนโปรแกรมให้เป็น virtual คือให้ต่อเติมได้นั้น ให้ดำเนินการดังนี้ คือ พิจารณากิจกรรมที่ชื่อเดียวกันว่า

- กิจกรรมใดที่ดำเนินกรรมวิธีแตกต่างกันให้ใช้คำว่า VIRTUAL ต่อท้ายในทุกระดับ ตามตัวอย่างที่ยกมาคือ Hide กับ Show (สำหรับ Init มีเรื่องที่จะกล่าวถึงต่อไป)

- กิจกรรมใดที่ดำเนินกรรมวิธีเหมือนกัน ตามตัวอย่างที่ยกมาคือ JumpTo ให้คงไว้ในบรรพบุรุษแรก คือ Point และไม่กำหนดในผู้สืบสกุลคือ Circle และต่อ ๆ ไปเช่น Rectangle

- กิจกรรมใดที่ต้องรันก่อนกิจกรรมอื่น ๆ ในออปเจ็คนั้น ๆ ให้เปลี่ยนชนิดของโปรแกรมย่อยจาก PROCEDURE เป็น CONSTRUCTOR ตามตัวอย่างที่ยกมาคือ Init ของทุกออปเจ็ค

เมื่อดำเนินการเช่นนี้คอมพายเลอร์จะจัดตาราง VMT (Virtual Method Table) 1 ตารางให้แก่แต่ละออปเจ็ค (ไม่ใช่แต่ละนิมิต) บรรจุพอยน์เตอร์ไปยังกิจกรรมที่เป็นเวอร์ชวล ทำให้สามารถต่อเติมออปเจ็คใหม่ได้ เพราะจะเป็นการลอกตารางของบรรพบุรุษมาแต่ให้พอยน์เตอร์ไปยังกิจกรรมที่เป็นเวอร์ชวลของตน

เช่นเมื่ออ้างถึง actorC.JumpTo จะหมายถึง JumpTo ใน Point เพราะใน Circle ไม่ได้กำหนดคำว่า JumpTo ทับ แต่ Hide และ Show ใน JumpTo จะเป็น Hide และ Show ของ Circle เพราะพอยน์เตอร์จะชี้มาที่นี่

สำหรับคำว่า CONSTRUCTOR เป็นแบบของ PROCEDURE แบบหนึ่ง ทำหน้าที่เชื่อมต่อนิมิตกับตาราง VMT จึงต้องให้เรียกใช้กิจกรรมนี้ก่อนกิจกรรมอื่นในนิมิตนั้น (ดังที่ตั้งชื่อไว้เป็น Init เพื่อเตือนใจ) และได้รวมการเตรียมค่าบุคลิกมาไว้ด้วยกัน เพราะเป็นกิจกรรมแรกที่จะต้องดำเนินการในนิมิตหนึ่ง ๆ เท่านั้น (ไม่ได้มีข้อกำหนดอื่นแต่อย่างใด)

ในการกำหนดนิมิตเราอาจกำหนดนิมิตหลายตัวให้เป็นแบบออปเจ็คเดียวกัน ในกรณีที่เป็นออปเจ็คแบบเวอร์ชวล นิมิตทั้งหมดนี้จะใช้ตาราง VMT ร่วมกัน แต่การให้ค่าแก่กันเช่นให้ค่านิมิต A แก่นิมิต B ไม่ถือว่าเป็นการเชื่อมต่อนิมิต B กับตาราง VMT จะต้องเชื่อมต่อนิมิต B กับตาราง VMT ก่อน (เช่น B.init) แล้วจึงจะให้ค่าต่อกันได้

เปรียบเทียบ

การกำหนดออปเจ็คในแบบสแตติกส์เหมาะสำหรับงานที่ตายตัวแล้ว ไม่มีความจำเป็นใดที่จะต้องเพิ่มออปเจ็คอีกดังเช่น ตัวหมากรุกดังกล่าวมา ทั้งนี้เพื่อประหยัดหน่วยความจำและรันได้เร็วกว่ากัน แต่ถ้าเห็นว่าอาจต้องต่อเติมออปเจ็คอีก และต้องการจะนำยูนิตออกเผยแพร่ในรูปของไฟล์ .TPU หรือต้องการสร้างไฟล์ .TPU ใหม่ในอนาคต ควรกำหนดออปเจ็คในแบบเวอร์ชวล

อีกเรื่องหนึ่งที่ต้องคิด

หากผมต้องการเขียนโปรแกรมเพื่อให้ตัวละครในยูนิต PICTØ และที่จะต่อเติมภายหลัง (เช่น Rectangle) เคลื่อนไหวไปตามคีย์ลูกศรทั้งสี่ ตามแต่ผู้ใช้จะกด จะเขียนโปรแกรมย่อยได้ดังรูปที่ 4

ในโปรแกรมหลักผมก็เพียงผ่านตัวละคร (นิมิต) ที่ต้องการ เช่น actorC ไปให้แก่ any พร้อมด้วยระยะทาง (เป็นจุดภาพเช่น 5) ที่จะให้กระโดดในแต่ละครั้งที่ผู้ใช้กดคีย์ลูกศรไปให้แก่ by ก็จะได้ผลตามที่กล่าวมา จนกว่าผู้ใช้จะกดคีย์ Enter

```
PROCEDURE  Draging(var  any : Point;dist : Integer);
VAR   dx,dy : Integer;  ch : Char;
BEGIN
 quit := False;
 REPEAT
  dx := 0;  dy := 0;
  repeat
    ch := ReadKey;
    if  ch = #0  then  ch := ReadKey;
  until  ch IN [up,down,left,right,Esc,CR];
  CASE  ch  OF
   up    : dec(dy,dist);    down  : inc(dy,dist);
   left  : dec(dx,dist);    right : inc(dx,dist);
   Esc CR : quit := True;
  END;  { case }
  if  not quit  then  any.JumpTo(GetX+dx, GetY+dy);
 UNTIL  quit;
END;
```

รูปที่ 4

C1

```
PROCEDURE  Point.Drag(dist : Integer);
VAR   dx,dy : Integer;  ch : Char;
BEGIN
 quit := False;
 REPEAT
  dx := 0;  dy := 0;
  repeat
    ch := ReadKey;
    if  ch = #0  then  ch := ReadKey;
  until  ch IN [up,down,left,right,Esc,CR];
  CASE  ch  OF
   up     : dec(dy,dist);     down  : inc(dy,dist);
   left   : dec(dx,dist);     right : inc(dx,dist);
   Esc,CR : quit := True;
  END;  { case }
  if  not quit  then  JumpTo(GetX+dx, GetY+dy);
 UNTIL  quit;
 if  ch = CR  then  quit := False;
END;


FUNCTION  Point.IsQuit : Boolean;
begin  IsQuit := Quit  end;

END.
```

```
  Point = OBJECT(Locate)
            quit : Boolean;
constructor  Init(fx,fy : Integer);
destructor   Fin;    VIRTUAL;
procedure    Hide;   VIRTUAL;
procedure    Show;   VIRTUAL;
procedure    JumpTo(nx,ny : Integer);
procedure    Drag(dist : Integer);
function     IsQuit : Boolean;
          END;
```

รูปที่ 5

แต่การที่เราเขียนโปรแกรมย่อยดำเนินการกับออปเจ็คไว้นอกออปเจ็คเช่นนี้ เป็นวิธีการที่ไม่ถูกต้องในการเขียนโปรแกรมแบบโอโอพี จะต้องให้เป็นกิจกรรมหนึ่งของออปเจ็ค

ปัญหาคือจะให้อยู่ในออปเจ็คใดดี คือ Locate, Point หรือ Circle ซึ่งหากให้อยู่ใน Locate ก็จะเรียกใช้ MoveTo ไม่ได้ หากให้อยู่ใน Circle ก็ไม่สามารถเคลื่อนย้าย Point ได้ ที่เหมาะที่สุดคือใน Point ดังแสดงในรูปที่ 5 (ขอให้แก้การกำหนดแบบข้อมูล Point ใน PICT∅.PAS ให้เป็นไปตามนี้ด้วยครับ)

สำหรับตัวโปรแกรมแท้ ๆ ของกิจกรรม Drag กับ IsQuit ตามรูป C1 ขอให้ป้อนไว้ในส่วน Implementation และควรนำออปเจ็ค Rectangle มาไว้ในยูนิตนี้ด้วย (ซึ่งของคุณยังคงเป็น PICT∅ แต่ของผมจะเป็น PICT∅ ดังจะเห็นในรูป M3 เพื่อการทำรูปประกอบเรื่องครับ)

ได้เขียนโปรแกรมใช้งานยูนิตนี้ไว้ในรูป M3 ซึ่งเพียงแต่ให้เลื่อนวงกลมไปมาด้วยคีย์ลูกศรทั้งสี่จนกว่าจะกดคีย์ Enter เท่านั้น (ยังไม่ได้แสดงการใช้กิจกรรม IsQuit)

ต่อไปได้เรื่อย ๆ

หากต่อไปผู้กำกับต้องการที่จะต่อออปเจ็คให้สามารถ

- กำหนดหน้าต่างบนจอภาพในกราฟิกโหมดได้
- บรรจุข้อความในหน้าต่างนี้ได้
- ให้ผู้ใช้สามารถเลื่อนหน้าต่างนี้ไปมาได้ด้วยคีย์ลูกศร

สิ่งที่ผู้กำกับจะต้องพิจารณาคือจะให้ออปเจ็คนี้สืบทอดจากออปเจ็คใดหากพิจารณาโดยผิวเผินจะเห็นว่าน่าจะให้เป็นทายาทของ Rectangle เพราะสามารถทำเป็นกรอบหน้าต่างได้ แต่การหายตัวและแสดงตัวของ Rectangle จะทำเฉพาะกรอบหน้าต่างเท่านั้นไม่รวมข้อความข้างใน

M3

```
{ EXP3.PAS  }

PROGRAM  ExampleThree;
uses   crt,graph,grp1,pict01;
VAR   actorC : Circle;

BEGIN
  OpenGraph;

  OutText('Use arrow keys to move the circle, end with <Enter>');
  actorC.Init(150,150,50);  actorC.Show;
  actorC.Drag(5);


readln;  CloseGraph
END.
```

D1

```
Gwindow = OBJECT(point)
  difx,dify,cpx,cpy : Integer;
  si : Word;
  old,newone : Pointer;
  constructor  Init(x1,y1,x2,y2 : Integer);
  destructor Fin;  VIRTUAL;
  procedure  Hide; VIRTUAL;
  procedure  Show; VIRTUAL;
  procedure  WindowOn;
  procedure  WindowOff;
  procedure  OutText(astr : String);
          END;

PROCEDURE  DispHeapStatus(astr : String);
```

ออปเจ็คใหม่นี้ซึ่งผมจะให้ชื่อว่า Gwindow จึงไม่จำเป็นต้องมีความสัมพันธ์กับ Rectangle นั่นก็คือ ควรให้สืบต่อจาก Point อีกเช่นกัน ดังการกำหนดแบบของออปเจ็ค Gwindow ในรูป D1 และมีตัวโปรแกรมอยู่ในรูป D2 ซึ่งมีรายละเอียดดังนี้คือ

บุคลิก

ได้กำหนดตัวแปรไว้เป็นบุคลิกดังนี้คือ

D2

```
CONSTRUCTOR  Gwindow.Init(x1,y1,x2,y2 : Integer);
BEGIN
  Point.Init(x1,y1);  cpx := 0;  cpy := 0;
  difx := x2-x1;  dify := y2-y1;
  si := ImageSize(x1,y1,x2,y2);
  GetMem(old,si);  GetMem(newone,si);
  GetImage(x1,y1,x2,y2,old^);
  Graph.Rectangle(x1,y1,x2,y2);
  GetImage(x1,y1,x2,y2,newone^);
END;


PROCEDURE  Gwindow.Hide;
var   xx,yy : Integer;
BEGIN
  xx := GetX;  yy := GetY;
  GetImage(xx,yy,xx+difx,yy+dify,newone^);
  PutImage(xx,yy,old^,CopyPut);
END;

PROCEDURE  Gwindow.Show;
var  xx,yy : Integer;
BEGIN
  xx := GetX;  yy := GetY;
  GetImage(xx,yy,xx+difx,yy+dify,old^);
  PutImage(xx,yy,newone^,CopyPut);
END;

PROCEDURE  Gwindow.WindowOn;
VAR   xx,yy : Integer;
BEGIN
  xx := GetX;  yy := GetY;
  SetViewPort(xx+2,yy+2,xx+difx-2,yy+dify-2,clipon);
END;

PROCEDURE  GWindow.WindowOff;
begin SetViewPort(0,0,GetMaxX,GetMaxY,clipon)  end;

PROCEDURE  Gwindow.OutText(astr : String);
VAR  hi,wid,numch,stt : Integer;
BEGIN
  WindowOn;
  hi := TextHeight('a');  wid := TextWidth('a');
  numch := (difx - 4) DIV wid;  stt := 1;
  WHILE  stt <= Length(astr)  DO
    begin  Graph.OutTextXY(cpx,cpy,Copy(astr,stt,numch));
           cpy := cpy + hi + 4;  stt := stt + numch   end;
  WindowOff;
END;

DESTRUCTOR  Gwindow.Fin;
BEGIN
  Hide;  FreeMem(newone,si);  FreeMem(old,si);
END;

PROCEDURE  DispHeapStatus(astr : String);
VAR    tstr : String;
BEGIN
  Str(MemAvail:6, tstr);
  OutText(astr + tstr);
  MoveTo(0,GetY + 12);
END;
```

- difx และ dify คือด้าน x และ y ของกรอบหน้าต่าง

- cpx และ cpy คือโคออร์ดิเนตของจุด CP ในหน้าต่าง เพื่ออ้างถึงในการนำภาพอักษรออกแสดงในหน้าต่าง

- si จะให้เก็บขนาดของหน้าต่างเป็นไบต์เพื่อใช้กับคำสั่ง GetMem ร่วมกับพอยน์เตอร์ old และ newone ในการเก็บภาพบนจอภาพ

Init

เพื่อดำเนินการเชื่อมต่อกับ VMT และให้เตรียมค่าต่าง ๆ ดังกล่าวมา จากค่ามุมบนซ้าย (x1, y1) และมุมล่างขวา (x2,y2) ของหน้าต่าง ในการนี้ให้เก็บภาพเดิมไว้ใน old ซึ่งในครั้งนี้จะเป็นภาพเปล่า กับให้เก็บภาพที่ได้ตีกรอบหน้าต่างแล้วไว้ใน newone ซึ่งจะเป็นหน้าต่างที่ว่างเปล่าเช่นกัน

โปรแกรมอื่น ๆ จะทำหน้าที่ตามที่กล่าวมาแล้ว และเป็นเรื่องของกราฟิกกับพอยน์เตอร์ จึงของดกล่าวในรายละเอียด ขอให้ป้อนโปรแกรมดังกล่าวมานี้ในยูนิต PICT∅

สำหรับโปรแกรมทดลองใช้งานดังแสดงในรูป M4

ลักษณะเด่นของโอโอพี

ผมคิดว่าเท่าที่กล่าวมานี้คงเพียงพอที่จะนำมาเป็นตัวอย่างประกอบคำอธิบายถึงลักษณะเด่นของการเขียนโปรแกรมในแบบโอโอพี (ซึ่งภาษาที่จะใช้จะต้องเอื้ออำนวยในเรื่องนี้ด้วย) กล่าวคือ

Encapsulation

หมายถึงการที่สามารถรวมข้อมูลและโปรแกรมย่อยเข้าด้วยกันเป็นแคปซูลเรียกเป็นออปเจ็คประกอบด้วยบุคลิก (คือข้อมูล) กับกิจกรรม (คือโปรแกรมย่อย) ทำให้

- เปรียบเสมือนว่าได้เก็บข้อมูลไว้ในที่ปกป้องอันหนึ่ง ซึ่งนอกจากตัวมันเองและผู้สืบสกุลแล้ว ผู้อื่น (โปร-

```
{ EXP4.PAS  }

PROGRAM  ExampleFour;
uses   crt,graph,grpl,pict02;
VAR    info,memo : Gwindow;

PROCEDURE   MainWindow;
begin  SetViewPort(0,0,GetMaxX,GetMaxY,clipon)  end;

BEGIN
  OpenGraph;

  info.Init(0,0,GetMaxX,50);
  memo.Init(200,100,300,150);  readln;

  WITH  memo  DO
    begin  WindowOn;
           OutText(' MEMORANDUM');
           OutText('  Turbo Pascal 5.5 gives you the power of ');
           OutText('Object-Oriented Programming');
    end;


  WITH  Info  DO
    begin
    WindowOn;
    OutText('You can move the window by arrow keys, quit with Enter');
    OutText('Exit from this program by press Esc');
    end;

  REPEAT
    memo.Drag(15);
    info.Drag(20);
  UNTIL  memo.IsQuit  or  info.IsQuit;


Graph.OutTextXY(10,30,'End of program press ENTER');

readln;  CloseGraph
END.
```

แกรมอื่น) จะแตะต้องไม่ได้ และการติดต่อกับข้อมูลก็จะต้องดำเนินการผ่านโปรแกรมย่อยในแคปซูลนั้น จะดำเนินการโดยตรงไม่ได้ เรียกได้ว่าเป็นแคปซูลหนึ่งเดียวกัน

- ผู้เขียนโปรแกรมในระดับหลัง ๆ (ผู้กำกับ) ไม่ต้องสนใจในข้อมูลว่ามีมากน้อยเพียงไรเป็นข้อมูลแบบใดบ้าง ตัดภาระในเรื่องข้อมูลโดยสิ้นเชิง ดังเช่นข้อมูลของ Circle กับ Gwindow แตกต่างกันด้วยจำนวนข้อมูลและแบบของข้อมูลอย่างมาก แต่ให้ทำงานในเรื่องเดียวกันได้เช่น JumpTo

Inheritance

สามารถให้สืบทอดกันได้ คือให้สามารถกำหนดออปเจ็คใหม่ สืบทอดบุคลิกและกิจกรรมของบรรพบุรุษได้ และด้วยการดัดแปลงเพียงเล็กน้อยก็สามารถให้ทำงานอื่นที่กว้างขวางออกไปได้

ดังเช่น Gwindow นำเรื่องบุคลิกของจุดโคออร์ดิเนตของ Point มาทำเป็นจุดมุมบนซ้าย และนำกิจกรรมเคลื่อนตัวคือ Drag ของ Point มาใช้ ด้วยการปรับปรุงกิจกรรม Hide กับ Show เท่านั้น ก็สามารถย้ายภาพทั้งหน้าต่างแทนที่จะเป็นจุดเพียงจุดเดียวได้

Polymorphism

คือความสามารถที่จะใช้สื่อความ (message) เดียวกันกับออปเจ็คต่าง ๆ ทำให้เกิดผลเช่นเดียวกัน แต่การปฏิบัติแตกต่างกัน ดังเช่นเมื่อใช้ JumpTo กับ Point หรือกับ Circle หรือกับ Gwindow ก็เพื่อจุดประสงค์จะย้ายตำแหน่งที่ของออปเจ็คดังกล่าว แต่การปฏิบัติแตกต่างกันมาก คือ เป็นการย้ายจุดเพียง 1 จุด หรือย้ายวงกลม 1 วง หรือย้ายหน้าต่างทั้งหน้าต่าง

ลักษณะของโปรแกรมในแบบโอโอพี ซึ่งเขียนด้วยเทอร์โบปาสคาลก็มีเพียงเท่านี้ครับ แต่ที่ยังขาดอยู่คือการกำหนดนิมิตในแบบพอยน์เตอร์ซึ่งต้องขอนำไปไว้ในฉบับต่อไป ส่วนเรื่องราวของการเขียนโปรแกรมในแบบโอโอพีโดยตรง จะหาดูได้ในหนังสือที่กล่าวถึงเรื่องนี้ ซึ่งส่วนมากจะยกตัวอย่างด้วยภาษา SmallTalk อันเป็นภาษาสำหรับการเขียนโปรแกรมในแบบโอโอพีโดยตรง

ต้องขอเวลาอีกสักระยะหนึ่งครับ ให้ผมสะสางเรื่องที่ค้างอยู่ก่อน จะนำมาถ่ายทอดด้วยภาษาปาสคาล